Programming Assignment 1 Checklist: WordNet

Frequently Asked Questions

Where are the textbook libraries `stdlib.jar` and `algs4.jar`? If you took *Algorithms Part I*, you should already have them installed. Otherwise, you can download them from the booksite. This page also contains links to all of the APIs.

Can I read the synset or hypernym file twice? No, file i/o is very expensive so please read each file only once and store it in an appropriate data structure.

Any advice on how to read in and parse the synset and hypernym data files? Use the `readLine()` method in our `In` library to read in the data one line at a time. Use the `split()` method in Java's `String` library to divide a line into fields. You can find an example using `split()` in Domain.java. Use `Integer.parseInt()` to convert string id numbers into integers.

It takes a very long time to read in the input files in DrJava. What should I do? Use the command line. DrJava incurs substantial overhead with input and output.

What assumption can I make about the digraph G passed to the SAP constructor? It can be any digraph, not necessarily a DAG.

What data structure(s) should I use to store the synsets, synset ids, and hypernyms? This part of the assignment is up to you. You must carefully select data structures to achieve the specified performance requirements.

Do I need to store the glosses? No, you won't use them on this assignment.

Can I use my own Digraph class? No, it must have the same API as our Digraph.java class; otherwise, you are changing the API to the SAP constructor (which takes a `Digraph` argument). Do not submit `Digraph.java`.

Should I re-implement breadth-first search in my SAP class? It depends. You are free to call the relevant methods in BreadthFirstDirectedPaths.java. However, you can improve performance by several orders of magnitude by implementing it yourself, optimizing it for repeated calls with the same digraph. (See the optional optimization section below.) If you choose to implement your own version of BFS, give it a different name, e.g., `DeluxeBFS.java`, and submit it.

I understand how to compute the `length(int v, int w)` method in time proportional to $E + V$ but my `length(Iterable<Integer> v, Iterable<Integer> w)` method takes time proportional to $a \times b \times (E + V)$, where a and b are the sizes of the two iterables. How can I improve it to be proportional to $E + V$? The key is using the constructor in `BreadthFirstDirectedPaths` that takes an iterable of sources instead of using a single source.

Is a vertex considered an ancestor of itself? Yes.

What is the root synset for the WordNet DAG?

```
38003,entity,that which is perceived or known or inferred to have its own distinct existence (living or nonliving)
```

Can a noun appear in more than one synset? Absolutely. It will appear once for each meaning that the noun has. For example, here are all of the entries in `synsets.txt` that include the noun `word`.

```
35532,discussion give-and-take word,an exchange of views on some topic; "we had a good discussion"; "we had a word or two about it"
56587,news intelligence tidings word,new information about specific and timely events; "they awaited news of the outcome"
59267,parole word word_of_honor,a promise; "he gave his word"
59465,password watchword word parole countersign,a secret word or phrase known only to a restricted group; "he forgot the password"
81575,word,a string of bits stored in computer memory; "large computers use words up to 64 bits long"
81576,word,a verbal command for action; "when I give the word, charge!"
81577,word,a brief statement; "he didn't say a word about it"
81578,word,a unit of language that native speakers can identify; "words are the blocks from which sentences are made"; "he hardly said te
```

Can a synset consist of exactly one noun? Yes. Moreover, there can be several different synsets that consist of the same noun.

```
62,Aberdeen,a town in western Washington
63,Aberdeen,a town in northeastern South Dakota
64,Aberdeen,a town in northeastern Maryland
65,Aberdeen,a city in northeastern Scotland on the North Sea
```

I'm an ontologist and I noticed that your `hypernyms.txt` file contains both *is-a* and *is-instance-of* relationships. Yes, you caught us. This ensures that every noun (except entity) has a hypernym. Here is an article on the subtle distinction.

How can I make the data type `SAP` immutable? You can (and should) save the associated digraph in an instance variable. However, because our `Digraph` data type is mutable, you must first make a defensive copy by calling the copy constructor.

What should `sap()`, `ancestor()`, or `outcast()` return if there is a tie for shortest ancestral path or outcast? The API does not specify, so you are free to return any such ancestor or outcast.

Do I need to throw exceptions explicitly with a throw statement? No, it's fine if they are thrown implicitly, e.g., you can rely on any method in Digraph.java to throw a `java.lang.IndexOutOfBoundsException` if passed a vertex argument outside of the prescribed range. A good API documents the requisite behavior for all possible arguments, but you should not need much extra code to deal with these corner cases.

Input, Output, and Testing

Some examples. Here are some interesting examples that you can use in your code.

- The SAP for `worm` and `bird` is `"animal animate_being beast brute creature fauna"` with a distance of 5. Other common ancestors of `worm` and `bird` with longer distances are: `"person individual someone somebody mortal soul"` and `"instrumentality instrumentation"`. The synsets that contain `worm` are:

  ```
  81679 worm
  81680 worm
  81681 worm
  81682 worm louse insect dirt_ball
  ```

  The synsets that contain `bird` are:

```
24306 bird
24307 bird fowl
25293 boo hoot Bronx_cheer hiss raspberry razzing razz snort bird
33764 dame doll wench skirt chick bird
70067 shuttlecock bird birdie shuttle
```

- The synset `municipality` has two paths to `region`.

```
municipality -> administrative_district -> district -> region
municipality -> populated_area -> geographic_area -> region
```

- The synsets `individual` and `edible_fruit` have several different paths to their common ancestor `physical_entity`.

```
individual -> organism being -> living_thing animate_thing -> whole unit -> object physical_object -> physical_entity
person individual someone somebody mortal soul -> causal_agent cause causal_agency -> physical_entity
edible_fruit -> garden_truck -> food solid_food -> solid -> matter -> physical_entity
edible_fruit -> fruit -> reproductive_structure -> plant_organ -> plant_part -> natural_object -> unit -> object -> physical_entity
```

- The following pairs of nouns are very far apart:

```
(distance = 23) white_marlin, mileage
(distance = 33) Black_Plague, black_marlin
(distance = 27) American_water_spaniel, histology
(distance = 29) Brown_Swiss, barrel_roll
```

- The following synset has many paths to `entity`.

```
Ambrose Saint_Ambrose St._Ambrose
```

- Also, we encourage you to use the small collection of sample files in the ftp directory.
- The number of nouns in `synsets.txt` is 119,188.

Possible progress steps

- Download wordnet-testing.zip. It contains sample input files for testing `SAP`, `WordNet`, and `Outcast`.
- Create the data type `SAP`. First, think carefully about designing a correct and efficient algorithm for computing the shortest ancestral path. Consult a staff member if you're unsure. In addition to the `digraph*.txt` files, design small DAGs to test and debug your code. Modularize by sharing common code.
- Read in and parse the files described in the assignment, `synsets.txt` and `hypernyms.txt`. Don't worry about storing the data in any data structures yet. Test that you are parsing the input correctly before proceeding.
- Create a data type `WordNet`. Divide the constructor into two (or more) subtasks (private methods).
  - Read in the `synsets.txt` file and build appropriate data structures. Record the number of synsets for later use.
  - Read in the `hypernyms.txt` file and build a `Digraph`. For this input, your digraph should have 82,192 vertices and 84,505 edges (but do not hardwire either of these numbers into your program because it must work for any valid input files).
- Implement the remaining `WordNet` methods.
- Implement `Outcast`.

Optional Optimizations

There are a few things you can do to speed up a sequence of SAP computations on the same digraph. Do not attempt to do this or any of your own invention without thoroughly testing your code. You can gain bonus points for correctly implementing some of these optimizations but you risk losing more points than you can gain if you introduce bugs that render your solution no longer correct.

- The bottleneck operation is re-initializing arrays of length V to perform the BFS computations. This must be done once for the first BFS computation, but if you keep track of which array entries change, you can reuse the same array from computation to computation (re-initializing only those entries that changed in the previous computation). This leads to a dramatic savings when only a small number of entries change (and this is the typical case for the wordnet digraph). Note that if you have any other loops that iterates through all of the vertices, then you must eliminate those to achieve a sublinear running time. (An alternative is to replace the arrays with symbol tables, where, in constant time, the constructor initializes the value associated with every key to be null.)
- If you run the two breadth-first searches from v and w in lockstep (alternating back and forth between exploring vertices in each of the two searches), then you can terminate the BFS from v (or w) as soon as the distance exceeds the length of the best ancestral path found so far.
- Implement a software cache of recently computed `length()` and `ancestor()` queries.

Enrichment

- This applet connects words by a chain of WordNet synonyms.
- This paper measures the semantic orientation of WordNet adjectives by computing their relative distance to "good" and "bad."